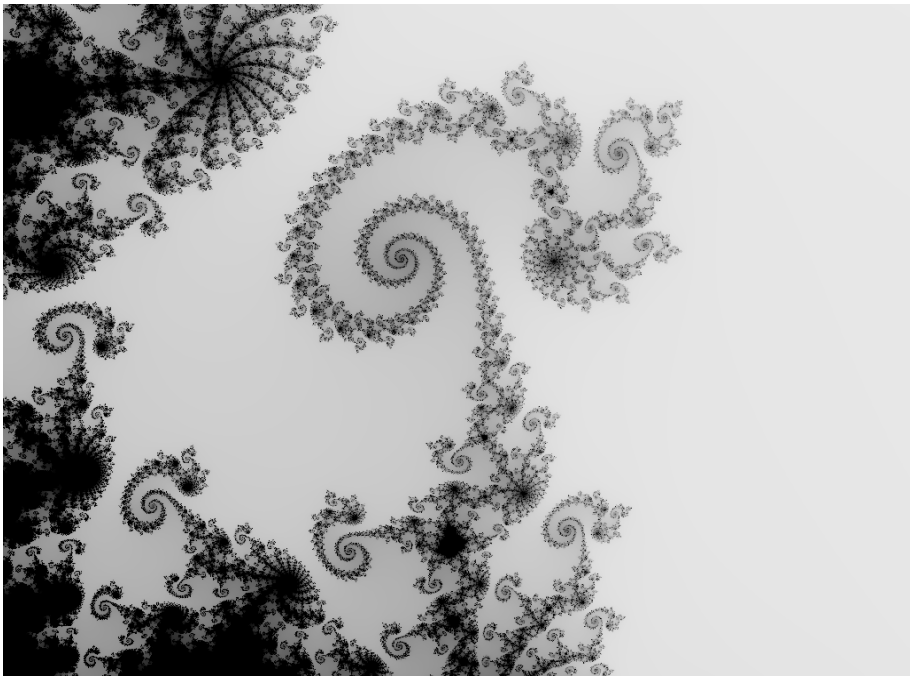
CS 789
Multiprocessor Programming

Optimizing the Sequential Mandelbrot Computation.

School of Computer Science
Howard Hughes College of Engineering
University of Nevada, Las Vegas



1 Sequential Mandelbrot Fractals

As we have talked about in class, when measuring (Amdahl's speedup) we use the following equation:

$$S(P) = \frac{t_s}{t_p}$$

The time for the sequential program t_s should be measured using the best known implementation of an algorithm.

This assignment is about getting a sequential version of the Mandelbrot Fractal calculation program to run fast. We will later on implement a parallel version.

I have implemented a version, you can find the code in Appendix A, and on line. In order for the program to generate nice pictures, you must provide it with the name of a `.map` file. A number of these files can be found in `~matt/CS789/Assignment1/maps`. My implementation can be found there too.

The version I have implemented produces a `.bmp` file with the constructed image. Such a file can be viewed using any picture viewer like `ee` or `xv`.

The picture on the front is created using the command: *Mandel 1024 768 -0.7801785714285 -0.7676785714285 -0.1279296875000 -0.1181640625000 maps/grey.map frac.bmp*. The typical starting positions for the well known Mandelbrot set are *-2.5 1.5 -1.5 1.5*.

Your assignment is to optimize this program as much as possible. You may rip things out of the code, or add others. However, it should take the same parameters, and produce the same output file. You may also get the compiler to turn on optimization if you think that might help.

Question 1: Copy my version of the program and compile it without optimizations. Use the coordinates of the picture on the front page, but generate a picture of size 2000 by 1500. This allows the code to run for long enough to get a good timing. Use the the 'time' command in the Unix shell to time the actual wall clock time the program took to execute. Do this 10 times, and calculate the average.

The above average will serve as the time that you have to 'beat' by optimizing the program.

Question 2: In a iterative fashion, make changes to your program until you cannot get it any faster. Each time you make a significant change, describe what the change was, and perform the measurements again (report the 10 times, and calculate the average time). Also calculate the improvement in percent that you have achieved compared to my initial version. Finally, you should note the options you passed to the compiler as well.

Question 3: What is the maximal improvement you have managed to obtain?

Due Date:

```
#include <stdio.h>
#include <stdlib.h>

struct complex {
  float real;
  float imag;
};

int cal_pixel(struct complex c) {
  int count, max;
  struct complex z;
  float temp, lengthsq;

  max = 256;
  z.real = 0;
  z.imag = 0;
  count =0;
  do {
    temp = z.real * z.real - z.imag * z.imag + c.real;
    z.imag = 2 * z.real * z.imag + c.imag;
    z.real = temp;
    lengthsq = z.real * z.real + z.imag * z.imag;
    count++;
  } while ((lengthsq < 4.0) && (count < max));
  return count;
}


int main(int argc, char *argv[]) {
  int disp_width, disp_height;
  float real_min, real_max, imag_min, imag_max, scale_real, scale_imag;
  FILE *f;
  struct complex c;
  int x,y,i;
  char str[256];

  int map[3][257];

  int** pic;

  if (argc != 9) {
    printf("Usage:\n Mandelbrot width height real-min real-max imag-min imag-max mapfil
    exit(1);
  }
```

```
/* Decode arguments */
disp_width = atoi(argv[1]);
disp_height = atoi(argv[2]);

real_min = atof(argv[3]);
real_max = atof(argv[4]);
imag_min = atof(argv[5]);
imag_max = atof(argv[6]);

/* Load the required colour map file */
f = fopen(argv[7],"r");
for(i=0;i<257;i++) {
  fgets(str,1000,f);
  sscanf(str,"%d %d %d",&(map[0][i]),&(map[1][i]),&(map[2][i]));
}
fclose(f);

/* Allocate space for the image */
pic = (int**)malloc(sizeof(int*)*disp_width);
for (x=0;x<disp_width;x++)
  pic[x] = (int*)malloc(sizeof(int)*disp_height);


/* Compute scaling factors */
scale_real = (real_max-real_min)/disp_width;
scale_imag = (imag_max-imag_min)/disp_height;


for (y=0; y<disp_height; y++) {
  for (x=0; x<disp_width; x++) {
    c.real = real_min + ((float) x * scale_real);
    c.imag = imag_min + ((float) y * scale_imag);
    i = cal_pixel(c);
    pic[x][y] = i;
  }
}

f = fopen(argv[8],"w");
fprintf(f,"P3\n%d %d\n255\n",disp_width,disp_height);

for (y=0; y<disp_height; y++) {
  for (x=0; x<disp_width; x++) {
    fprintf(f,"%d %d %d ",map[0][pic[x][y]],map[1][pic[x][y]],map[2][pic[x][y]]);
  }
  fprintf(f,"\n");
```

```
    }
    fclose(f);
}
```